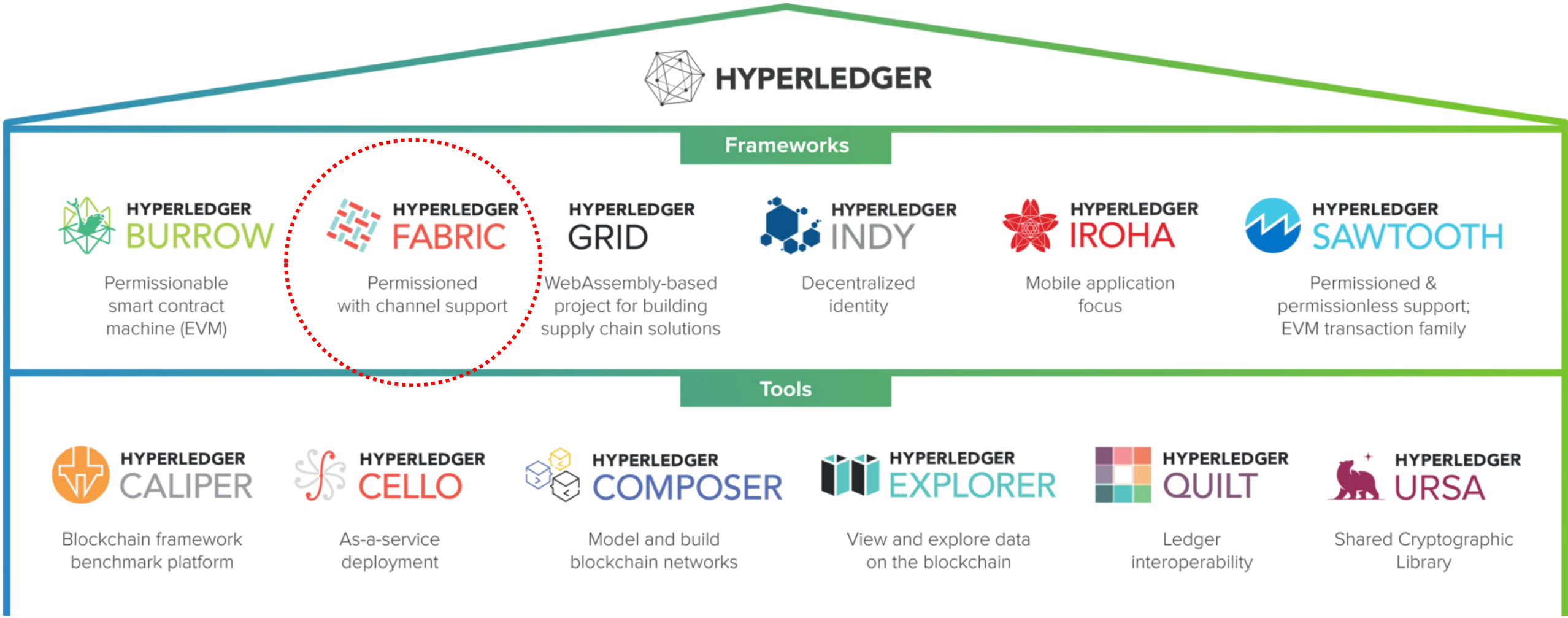
Hyperledger fabric (HLF)

background

- Problems of public blockchains
  - too slow
  - No identity information
- Enterprise use cases require …
  - high throughput
  - low delay
  - Identities should be known
    - Know-Your-Customer (KYC) and Anti-Money Laundering (AML)
- The Linux Foundation announced the creation of the Hyperledger
  - an umbrella project of open source blockchains and related tools
  - IBM, Intel, SAP, ….

Hyperledger project



Requirements for enterprise blockchain apps

- Participants must be identified/identifiable
- Networks need to be *permissioned*
- High transaction throughput performance
- Low latency of transaction confirmation
- Privacy and confidentiality of transactions and data pertaining to business transactions

Hyperledger Fabric (HLF) overview

- a highly modular and configurable architecture
- the first distributed ledger platform to support smart contracts in general-purpose programming languages such as Java, Go and Node.js
- Permissioned: the participants are known to each other
- pluggable consensus protocols for particular use cases and trust models
  - crash fault-tolerant (CFT) vs byzantine fault tolerant (BFT)
  - If f faulty nodes, Paxos (2f+1 nodes) vs PBFT (3f+1 nodes)
- do not require a native cryptocurrency to incentivize costly mining or to fuel smart contract execution
- Better TX processing and TX confirmation latency
- Privacy and confidentiality

modularity

- pluggable *ordering service* establishes consensus on the order of transactions and then broadcasts blocks to peers
- Pluggable *membership service provider* is responsible for associating entities in the network with cryptographic identities
- optional *peer-to-peer gossip service* disseminates the blocks output by ordering service to other peers
- Smart contracts ("chaincodes") run within a container environment (e.g. Docker) for isolation. They do not have direct access to the ledger state
- The ledger can be configured to support a variety of DBMSs
- pluggable endorsement and validation policy enforcement that can be independently configured per application

no "one blockchain to rule them all"

Permissionless vs permissioned

- Anyone can participate
- Anonymous
- No trust
- Cryptocurrency for incentive
  - E.g. proof of work

- Identified participants
- Some degree of trust
  - may not fully trust each other
- provides a way to secure the interactions among a group of entities
- Can use crash fault tolerant (CFT) or byzantine fault tolerant (BFT) consensus than mining

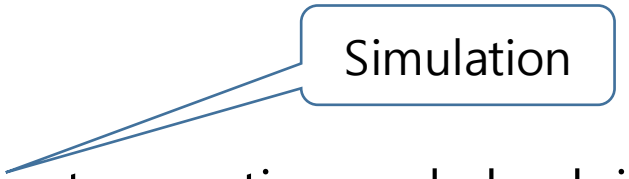
Smart contract (i.e. chaincode)

- a trusted distributed application that gains its security/trust from the blockchain and the underlying consensus
- The biz logic of a blockchain app
- many smart contracts run concurrently in the network
- they may be deployed dynamically (in many cases by anyone)
- application code should be treated as untrusted, potentially even malicious
- Order-execute vs execute-order-validate

Order-execute (in Bitcoin, Ethereum)

- Two phases
  - validates and orders transactions then propagates them to all peers
  - each peer then executes the transactions sequentially.
- must be deterministic; otherwise, consensus might never be reached.
- a non-standard, or domain-specific language (such as Solidity)
- As all transactions are executed sequentially by all nodes, performance and scale is limited
- complex measures be taken to protect the overall system from potentially malicious contracts

execute-order-validate (in Hyperledger fabric)



- 3 phases
  - *execute* a transaction and check its correctness, thereby endorsing it,
  - *order* transactions via a (pluggable) consensus protocol, and
  - *validate* transactions against an application-specific endorsement policy before committing them to the ledger
- an application-specific endorsement policy specifies which peers, or how many of them, need to vouch for the correct execution of a given smart contract
  - each transaction need only be executed (endorsed) by the subset of the peer nodes necessary to satisfy the transaction's endorsement policy -> parallelism
- The 1st phase eliminates any non-determinism, as inconsistent results can be filtered out before ordering -> enables use of standard programming languages (node.js, Go, Java)

Privacy and confidentiality

- in a public, permissionless blockchain network that leverages PoW for its consensus model, transactions are executed on every node -> no confidentiality
- many businesses/enterprise use cases require confidentiality
- Possible solutions
  - Encryption may be broken with enough time and resources
  - ZKP requires considerable time/resources
- Channel architecture
  - “channel” between the subset of participants that should be granted visibility to a particular set of transactions
  - nodes that participate in a channel have access to the smart contract (chaincode) and data transacted
  - One ledger per channel

Pluggable consensus

- The ordering of transactions is delegated to a modular component for consensus that is logically decoupled from the peers that execute transactions and maintain the ledger
- Since consensus is modular, its implementation can be tailored to the trust assumption of a particular deployment or solution like CFT and BFT
- SOLO, Kafka, PBFT
- Multiple ordering services can coexist

Performance and scalability

- Factors
  - Network size, TX size, block size,...
- Performance metrics
  - Read Latency
  - Read Throughput
  - Transaction Latency
  - Transaction Throughput

HLF concepts

- Identity
- Membership
  - Membership service provider (MSP)
- Peer
- Channel
- Ledger: world state + blockchain

HLF identity

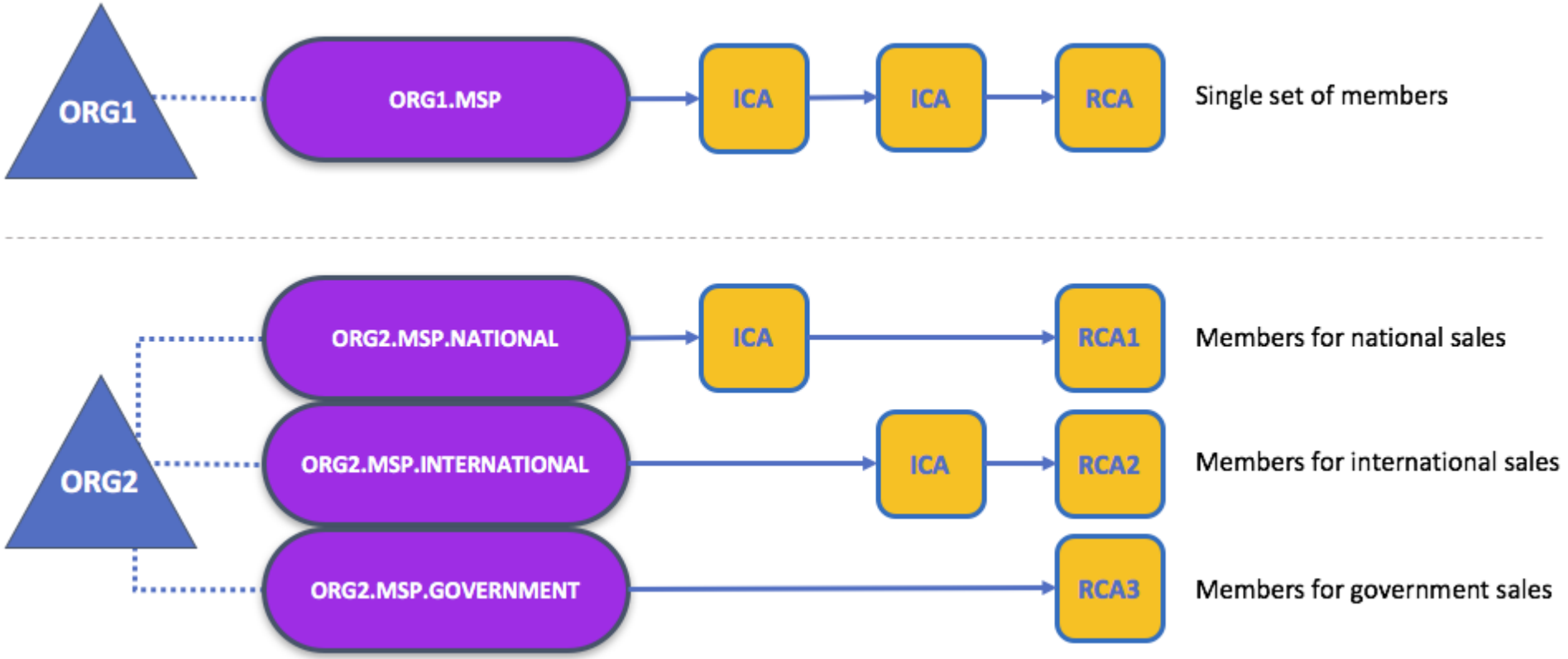
- The different actors in a blockchain network include peers, orderers, client applications, administrators and more
- Each actor has a digital identity in an X.509 certificate
- Identities determine the exact permissions over resources and access to information that actors have in a blockchain network
- A principal: the union of an identity and the associated attributes -> determines permissions
  - Include properties like actor's organization, role, specific identity
- PKI assures the binding between the principal and its public key
- PKI provides a list of identities, and an MSP says which of these are members of a given organization that participates in the network
- Membership service providers (MSPs) turn verifiable identities into the members of a blockchain network.

HLF membership

- MSP identifies which CAs are trusted to define the members of a trust domain, e.g., an organization, either by listing the identities of their members, or by identifying which CAs are authorized to issue valid identities for their members
- MSP can identify specific roles of an actor and define access privileges
- The configuration of an MSP is advertised to all the channels where members of the corresponding organization participate (in the form of a channel MSP). In addition to the channel MSP, peers, orderers, and clients also maintain a local MSP to authenticate member messages outside the context of a channel and to define the permissions over a particular component

HLF: mapping MSPs to organizations

- An organization is a managed group of members
  - Organizations may manage their members under a single MSP
  - An organization is often divided up into multiple organizational units (OUs), each of which has a certain set of responsibilities under different MSPs



ICA: intermediate CA, RCA: root CA

HLF: MSPs

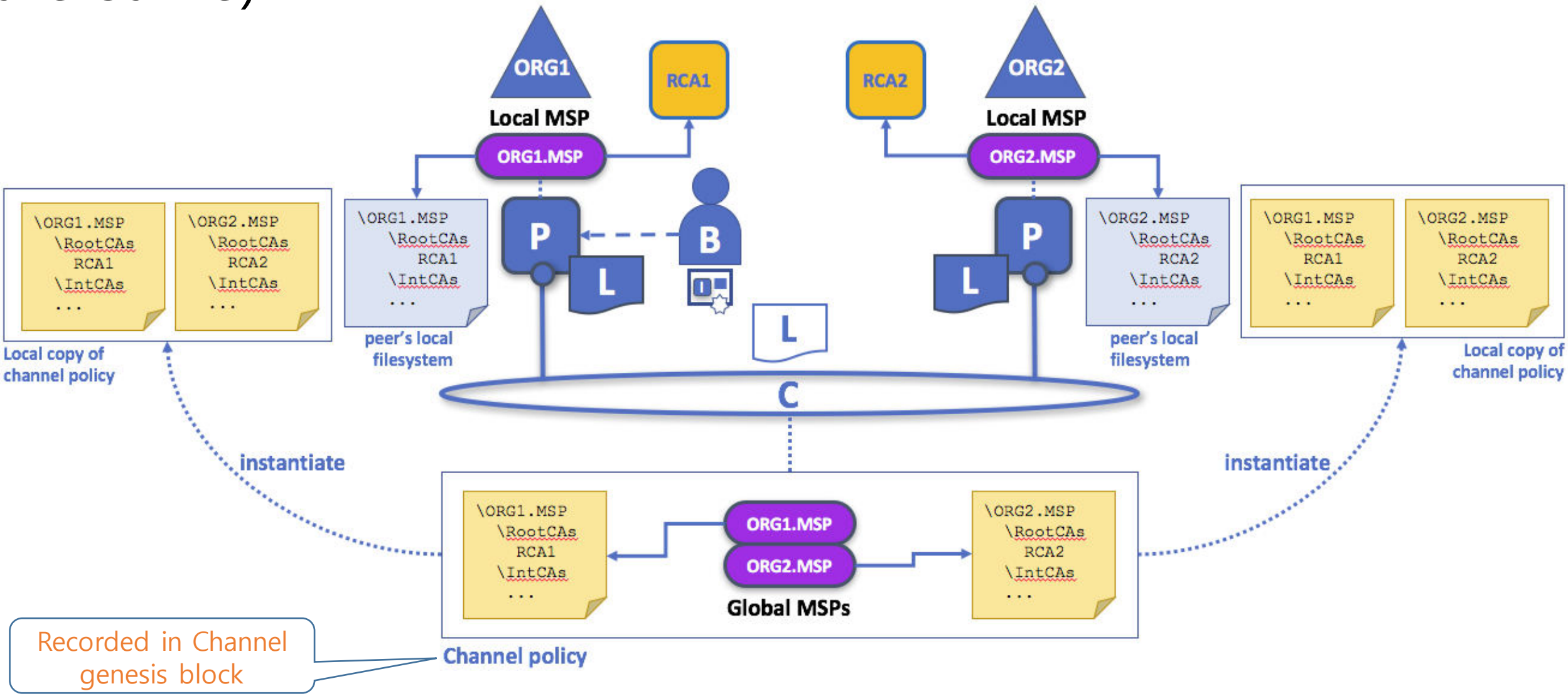
Local MSP

- defined for clients (users) and for nodes (peers and orderers)
- Every node and user must have a local MSP defined
- Node local MSPs define the permissions for that node
- local MSPs of the users allow the user side to authenticate itself in its transactions as a member of a channel, or as the owner of a specific role into the system (e.g. an org admin)
- Only stored on the file system of the node or user
- only one local MSP per node or user

Channel MSP

- define administrative and participatory rights at the channel level
- Peers/orderers on a channel will all share the same view of channel MSPs
  - Can authenticate the channel participants
- if an organization wishes to join the channel, an MSP for the organization's members would need to be included in the channel configuration
- is instantiated on the file system of every node in the channel and kept synchronized via consensus.

Local and channel MSPs have different scopes (functions are same)



1. Admin B connects to the peer with an identity issued by RCA1 and stored in their local MSP
2. When B tries to install a smart contract on the peer, the peer checks its local MSP (ORG1.MSP) to verify that the identity of B is indeed a member of ORG1
3. A successful verification will allow the install to complete
4. B wishes to instantiate the smart contract on the channel
5. Since this is a channel operation, all organizations on the channel must agree to it.
6. The peer must check the MSPs of the channel before it can commit the command

B: blockchain admin
P: peer
L: ledger
C: channel
RCA1: root CA1
ORG1: organization
ORG1.MSP: MSP of ORG1

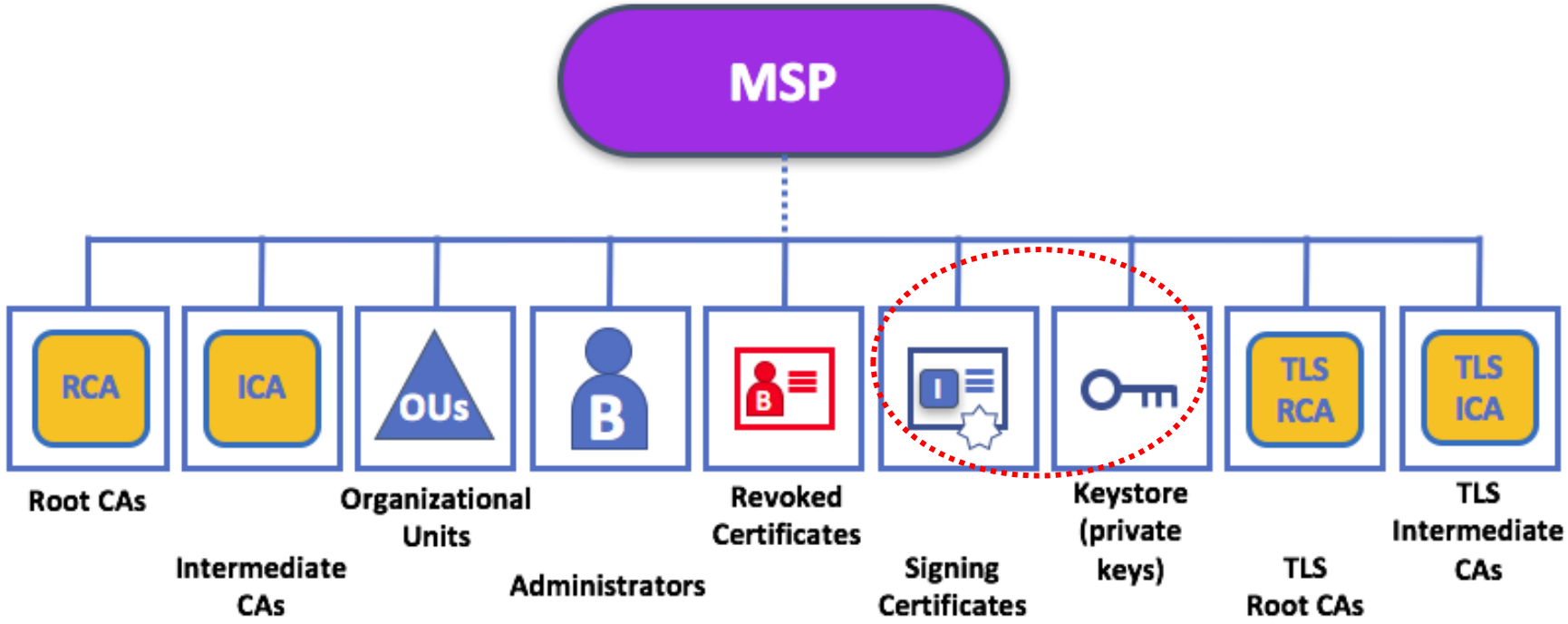
MSP levels

- Network MSP
  - configuration of a network defines who are the members in the network
    - Define the MSPs of the participant organizations
    - Define which of these members are authorized to perform administrative tasks (e.g., creating a channel)
- Channel MSP
  - It is important for a channel to maintain the MSPs of its members separately
  - A channel provides private communications between a particular set of organizations which in turn have administrative control over it
  - Channel policies interpreted in the context of that channel's MSPs define who has ability to participate in certain action on the channel, e.g., adding organizations, or instantiating chaincodes.

MSP levels

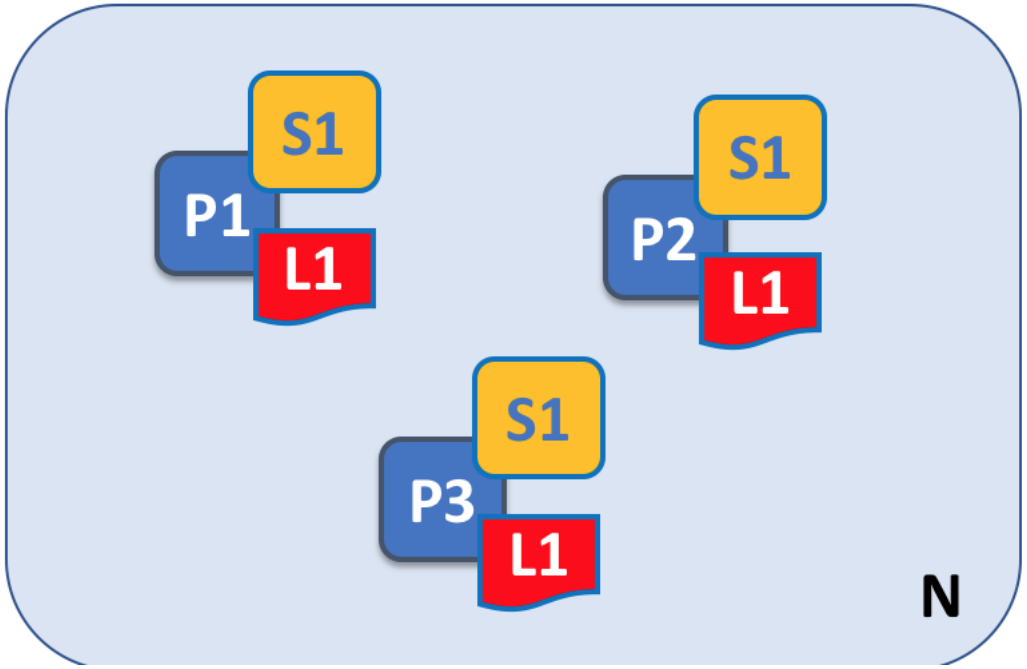
- peer MSP
  - This local MSP is defined on the file system of each peer and there is a single MSP instance for each peer.
  - it performs exactly the same function as channel MSPs with the restriction that it only applies to the peer
    - E.g. installation of a chaincode on the peer
- orderer MSP
  - Like a peer MSP, an orderer local MSP is also defined on the file system of the node and only applies to that node.
  - orderers are also owned by a single organization and therefore have a single MSP to list the actors or nodes it trusts

MSP structure



- A local MSP is stored on a local filesystem
- Admin: a list of identities that define the actors who have the role of administrators for this organization
  - Even if an actor has the role of an administrator, it doesn't mean that they can administer particular resources
  - actual power a given identity has with respect to administering the system is determined by the channel policies
- Certificate: only one X.509 certificate for the node
- Keystore: exactly one private key
  - Node's signing key

peers



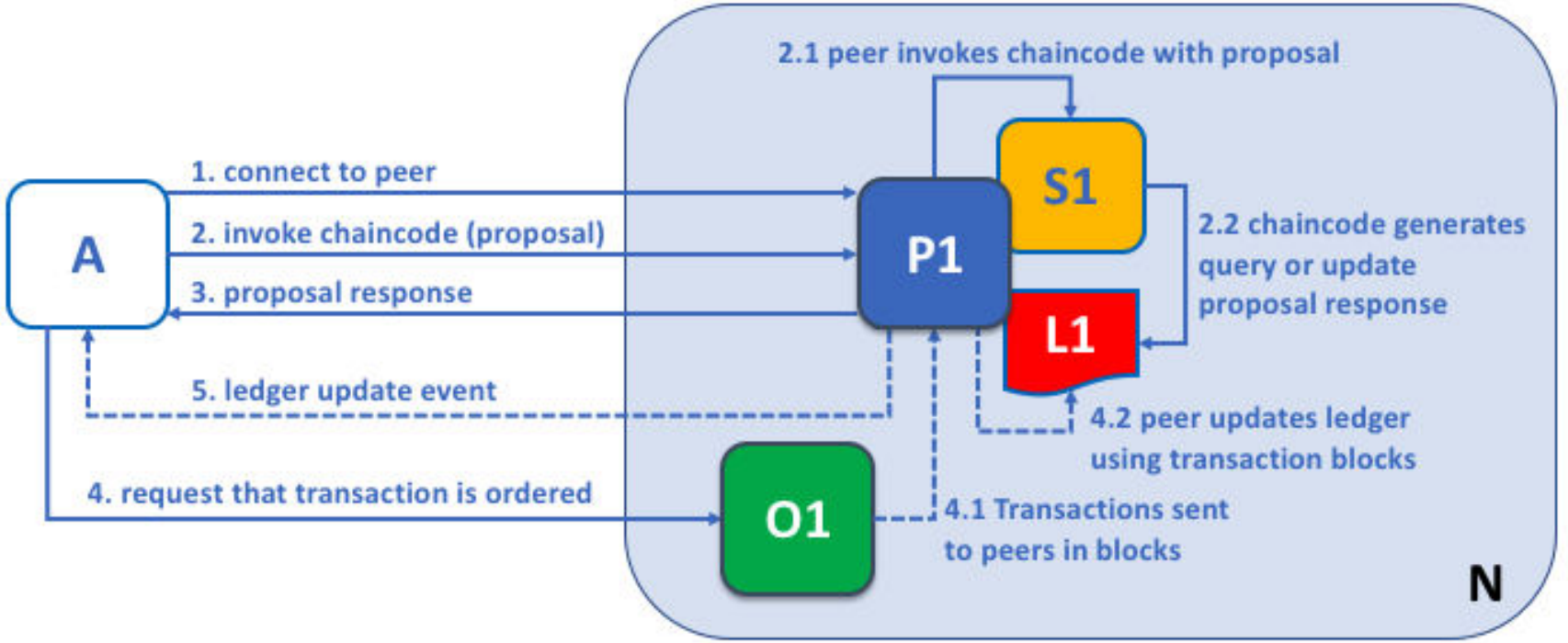
N	Blockchain network
P	Peer node
S	Smart contract (aka chaincode)
L	Ledger

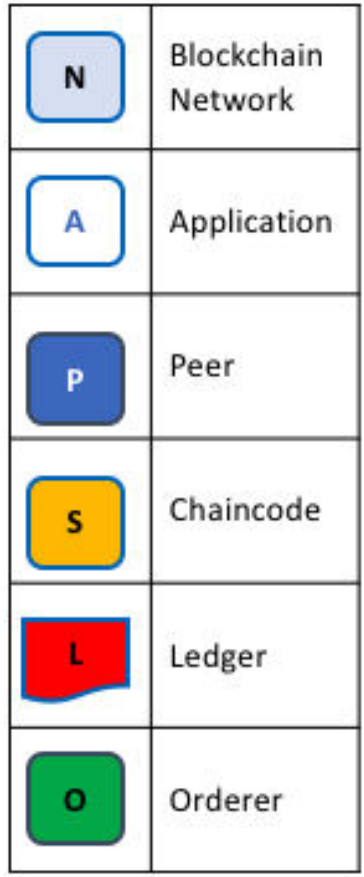
- A blockchain network is comprised primarily of a set of *peer nodes* (or, simply, *peers*)
- Peers are a fundamental element of the network because they host ledgers and smart contracts
- Smart contracts and ledgers are used to encapsulate the shared *processes* and shared *information* in a network, respectively

A peer hosts ledgers and chaincodes

- the peer actually hosts *instances* of the ledger, and *instances* of chaincode
- applications and administrators must interact with a peer if they want to access these resources
- A peer is able to host more than one ledger, which is helpful because it allows for a flexible system design
- A peer will have at least one chaincode installed on it which can query or update the peer's ledger instances

application accesses ledgers & chaincodes





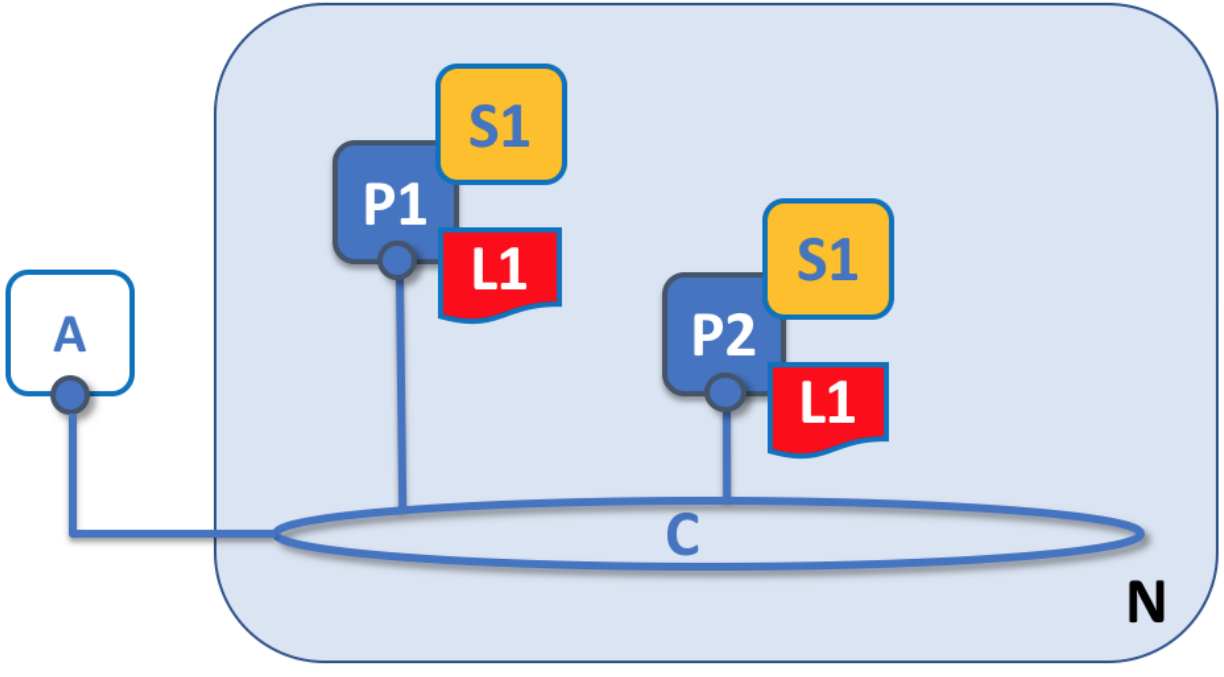
- Applications always connect to peers when they need to access ledgers and chaincodes: 3 steps for query, 5 steps for update
- Through a peer connection, applications can execute chaincodes to query or update a ledger
- A peer can return the results of a query to an application immediately since all of the information required to satisfy the query is in the peer's local copy of the ledger

In case of ledger update

- An update transaction starts in the same way as a query transaction, but has two extra steps
- an individual peer cannot perform a ledger update at the time of update request
- other peers must first agree to the change — a process called consensus
- peers return to the application a proposed update — one that this peer would apply subject to other peers' prior agreement
- step four — requires that applications send an appropriate set of matching proposed updates to the entire network of peers as a transaction for commitment to their respective ledgers
- This is achieved by the application using an orderer to package transactions into blocks, and distribute them to the entire network of peers

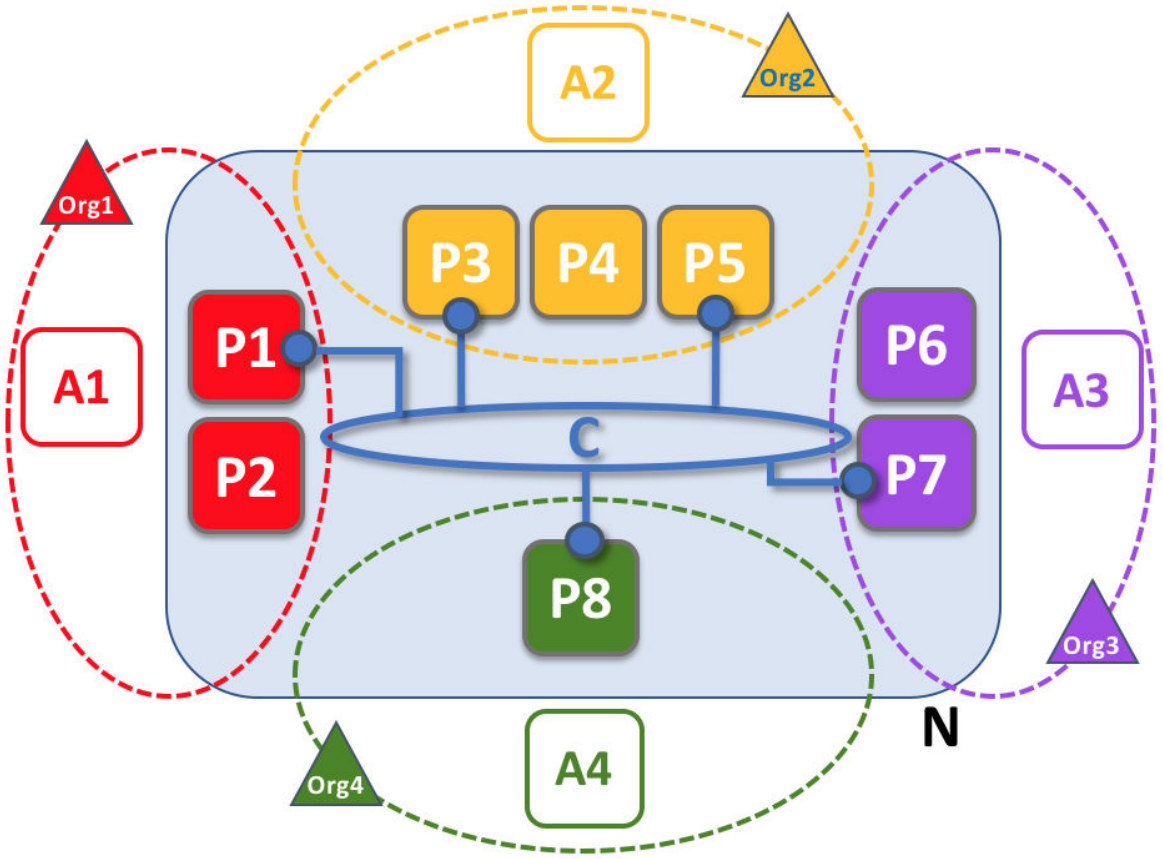
channel

- a mechanism by which a set of components within a blockchain network can communicate and transact *privately*
- By joining a channel, they agree to collaborate to collectively share and manage identical copies of the ledger associated with that channel
- Channels allow a specific set of peers and applications to communicate with each other within a blockchain network. In this example, application A can communicate directly with peers P1 and P2 using channel C.



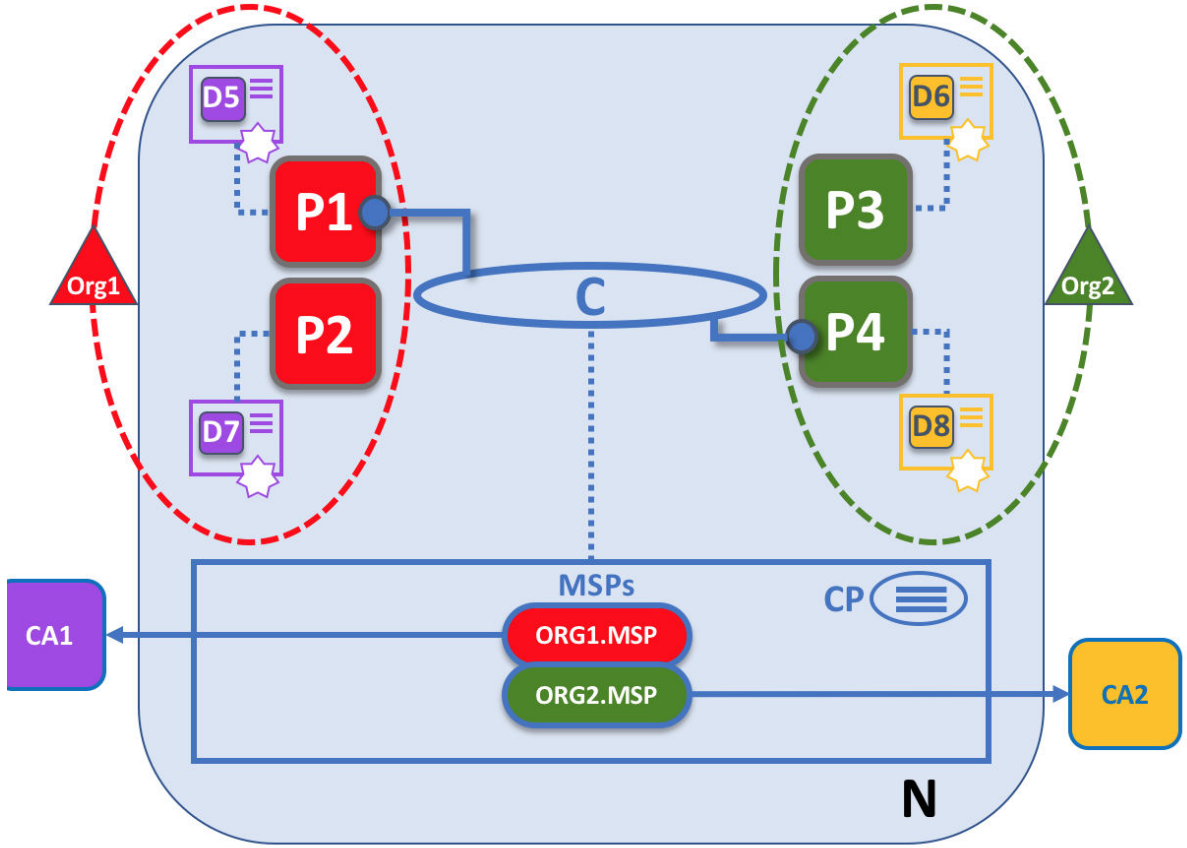
N	Blockchain Network	L	Ledger
C	Channel	A	Application
P	Peer	PA (C)	Principal PA (e.g. A, P1) communicates via channel C.
S	Chaincode		

Peers in a blockchain network with multiple organizations. The blockchain network is built up from the peers owned and contributed by the different organizations. In this example, we see four organizations contributing eight peers to form a network. The channel C connects five of these peers in the network N — P1, P3, P5, P7 and P8. The other peers owned by these organizations have not been joined to this channel, but are typically joined to at least one other channel. Applications that have been developed by a particular organization will connect to their own organization's peers as well as those of different organizations. Again, for simplicity, an orderer node is not shown in this diagram.



N	Blockchain Network	L	Ledger
C	Channel	A	Application
P	Peer	PA, C	Principal PA (e.g. A1, P5) communicates via channel C.
		Org	Organization
A1, R, P1, P2			Organization R owns application A1 and peers P1, P2.

When a peer connects to a channel, its certificate identifies its owning organization via a channel MSP. In this example, P1 and P2 have identities issued by CA1. Channel C determines from a policy in its channel config. that identities from CA1 should be associated with Org1 using ORG1.MSP. Similarly, P3 and P4 are identified by ORG2.MSP as being part of Org2.



N	Blockchain Network	P	Peer
C	Channel	Org	Organization
I	Identity	PA, C	Principal PA (e.g. P1,P4) communicates via channel C.
CP	Channel policy		
CA	Certificate Authority	MSP	Membership Service Provider
A1, R, P1, P2	Organization R owns application A1 and peers P1, P2.		
C, CP	Channel C subject to policy CP.	MSP1, MSP2, CP	Channel policy CP contains MSPs: MSP1 and MSP2.
MSP1, CA1	MSP1 selects the Certificate Authority CA1 to provide certificates for it.		

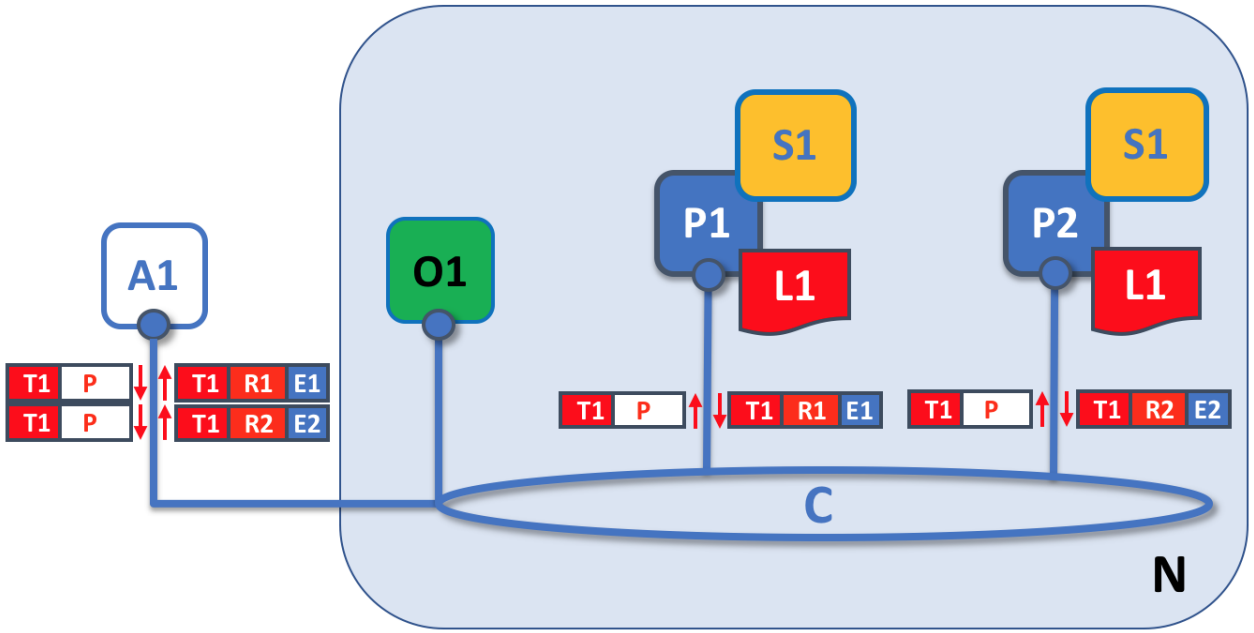
PA, D	Principal P has identity D

orderer

- the mechanism by which applications and peers interact with each other to ensure that every peer's ledger is kept consistent is mediated by special nodes called *orderers*
- updating the ledger requires the consent of other peers in the network – consensus
- 3-phase (i.e. 5 steps previously) process
  1. applications work with a subset of *endorsing peers*, each of which provides an endorsement of the proposed ledger update to the application, but do not apply the proposed update to their copy of the ledger
  2. these separate endorsements are collected together as transactions and packaged into blocks
  3. these blocks are distributed back to every peer where each transaction is validated before being applied to that peer's copy of the ledger

Update phase 1: proposal

- applications generate a transaction proposal which they send to each of the required set of peers for endorsement. Each of these *endorsing peers* then independently executes a chaincode using the transaction proposal to generate a transaction proposal response. It does not apply this update to the ledger, but rather simply signs it and returns it to the application
- Chaincode is run only in phase 1



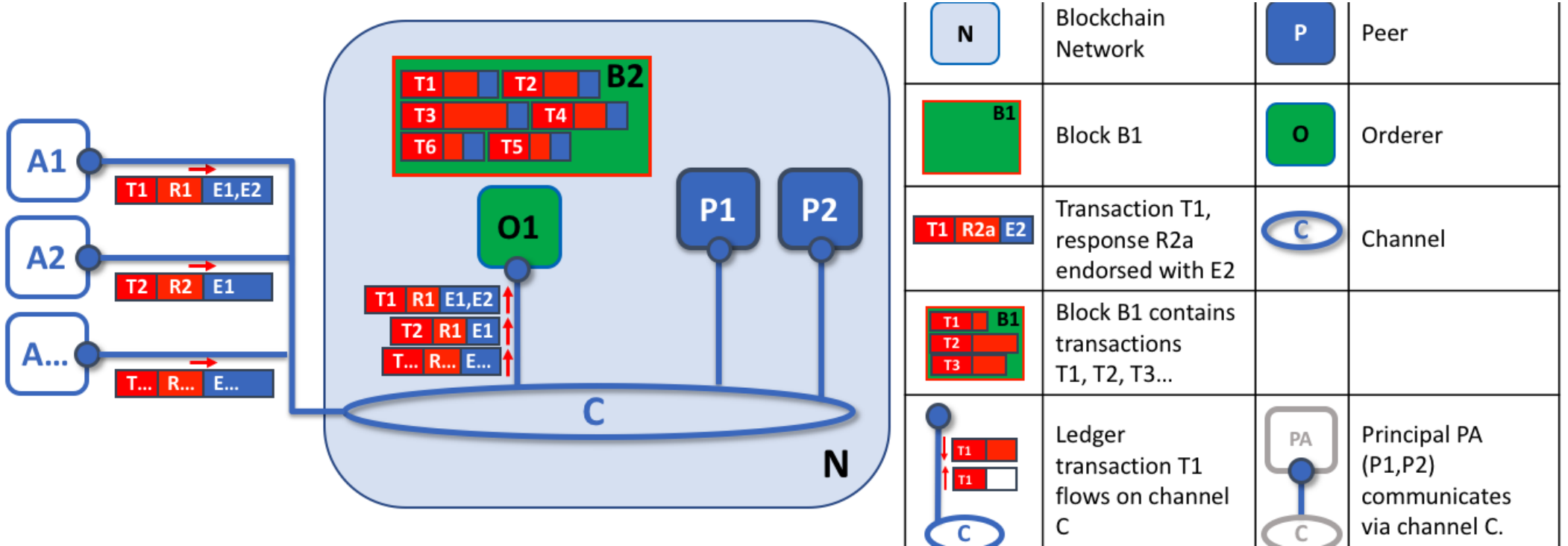
N	Blockchain Network	S	Chaincode
C	Channel	O	Orderer
P	Peer	L	Ledger
T1 P	Transaction T proposal P	T1 R2 E2	Transaction T1, response R2 endorsed with E2
T T C	Ledger transaction T1 flows on channel C	PA C	Principal PA (P1,P2) communicates via channel C.

After phase 1: what if results are different?

- different peers can return different and therefore inconsistent transaction responses to the application *for the same transaction proposal*.
- Maybe the result was generated at different times on different peers with ledgers at different states
- Less likely, but much more seriously, results might be different because the chaincode is *non-deterministic*.
  - Non-determinism is a serious problem as inconsistent results cannot be applied to ledgers.
- Application can discard inconsistent responses
- Consensus is needed

Phase 2: packaging

- The orderer receives transactions containing endorsed transaction proposal responses from many applications on a particular channel
- It orders each transaction relative to other transactions, and packages batches of transactions into blocks ready for distribution back to all peers connected to the orderer, including the original endorsing peers
- Once an orderer has generated a block of the desired size, or after a maximum elapsed time, it will be sent to all peers connected to it on a particular channel.

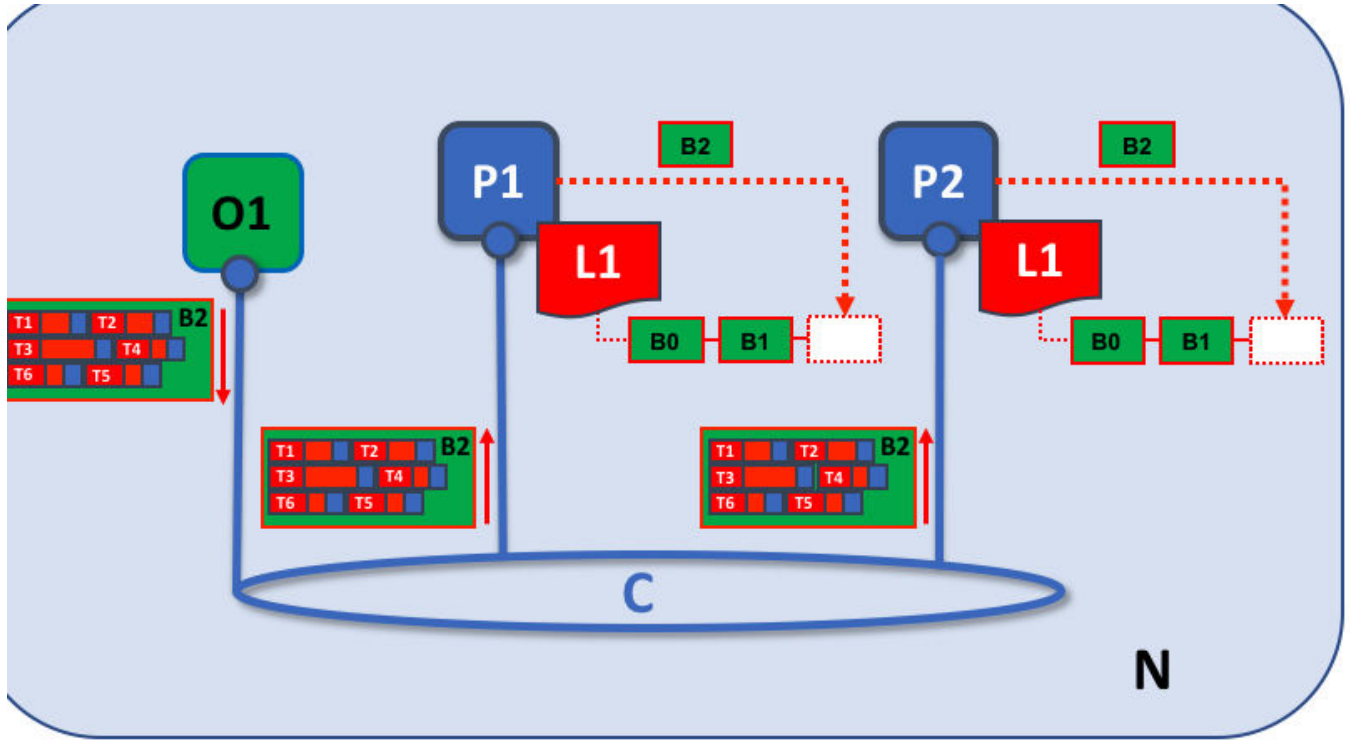


Ordering of TXs in phase 2

- the sequencing of transactions in a block is not necessarily the same as the order of arrival of transactions at the orderer
- Strict order: TXs can be packaged in any order into a block, and it's this sequence that becomes the order of execution
- the blocks generated by a collection of orderers are said to be "final" because once a transaction has been written to a block, its position in the ledger is immutably assured – no ledger fork
- At the end of phase 2, we see that orderers have been responsible for the simple but vital processes of collecting proposed transaction updates, ordering them, packaging them into blocks, ready for distribution
- Multiple orderers?
- orderers do not host the ledger and chaincodes

Phase 3: validation

- the distribution and subsequent validation of blocks from the orderer to the peers, where they can be applied to the ledger
- at each peer, every TX within a block is validated to ensure that it has been consistently endorsed by all relevant organizations before it is applied to the ledger
- not every peer needs to be directly connected to an orderer — peers can cascade blocks to other peers using the gossip protocol



N	Blockchain Network	P	Peer
C	Channel	O	Orderer
L	Ledger	B	Block B
L1 B0 B1	Ledger L1 has blockchain with blocks B0, B1	B1 T1 T2 T3	Block B1 contains transactions T1, T2, T3...
B1 C	Block B1 flows on channel C	PA C	Principal PA (P1, P2) communicates via channel C.

More on validation in phase 3

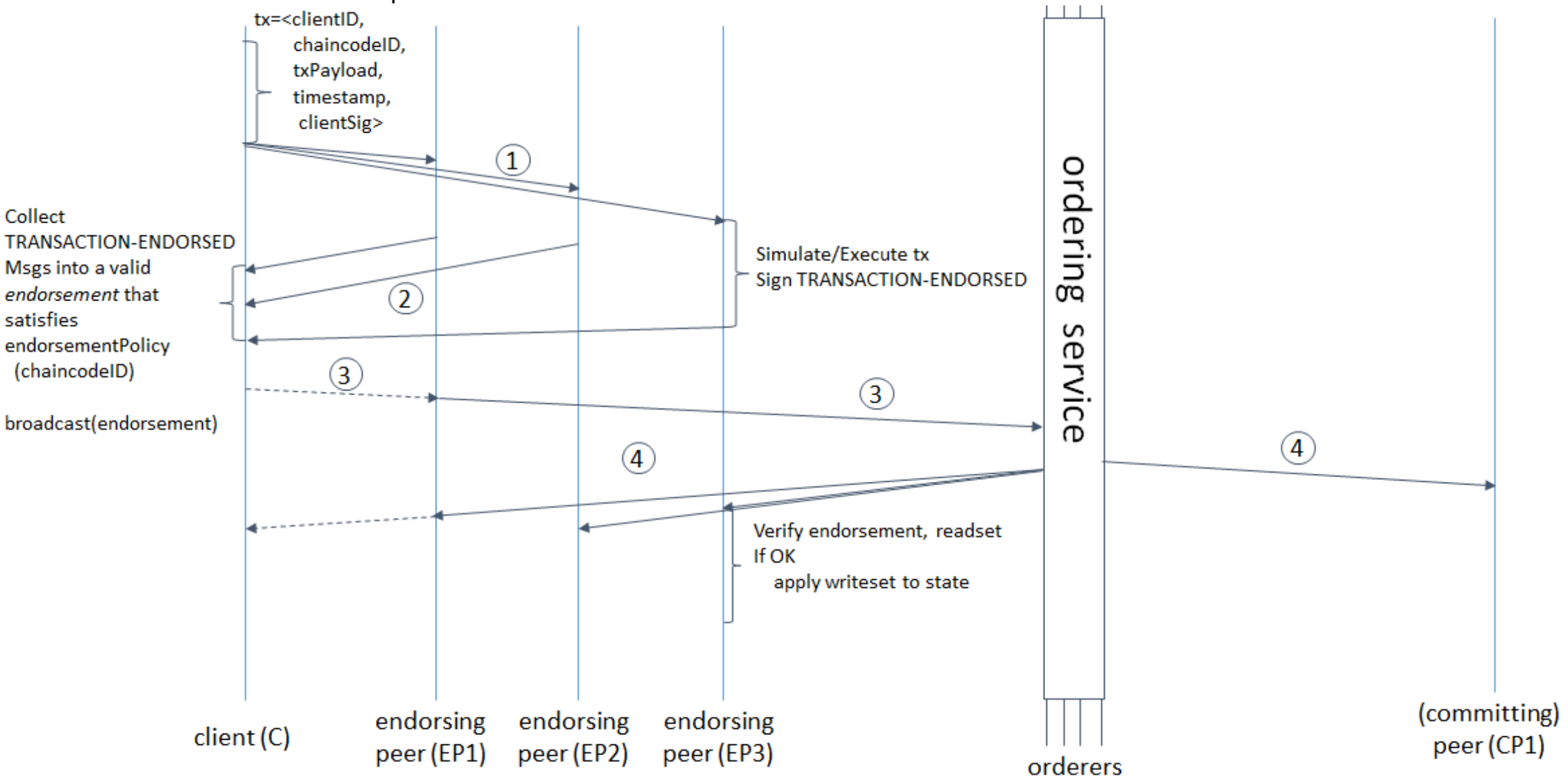
- for every TX, each peer will verify that the TX has been endorsed by the required organizations according to the *endorsement policy* of the chaincode which generated the transaction
- this validation is different than the endorsement check in phase 1, where it is the application that receives the response from endorsing peers and makes the decision to send the proposal TXs
- a peer must perform a ledger consistency check to verify that the current state of the ledger is compatible with the state of the ledger when the proposed update was generated
- chaincode is available only on endorsing nodes (running only in phase 1)
  - Help keep chaincode confidential
  - Chaincode stages: (packaging) – install – instantiate – invoke
- Output of chaincodes is shared with every peer in the channel
- Everytime a block is committed, the peer generates an event

① <PROPOSE,tx,[anchor]>

- tx=<clientID,chaincodeID,txPayload,timestamp,clientSig>
  - Invoke TX: txPayload = <operation, metadata>
  - Deploy TX: txPayload = <source, metadata, policies>
- Anchor: read version dependencies

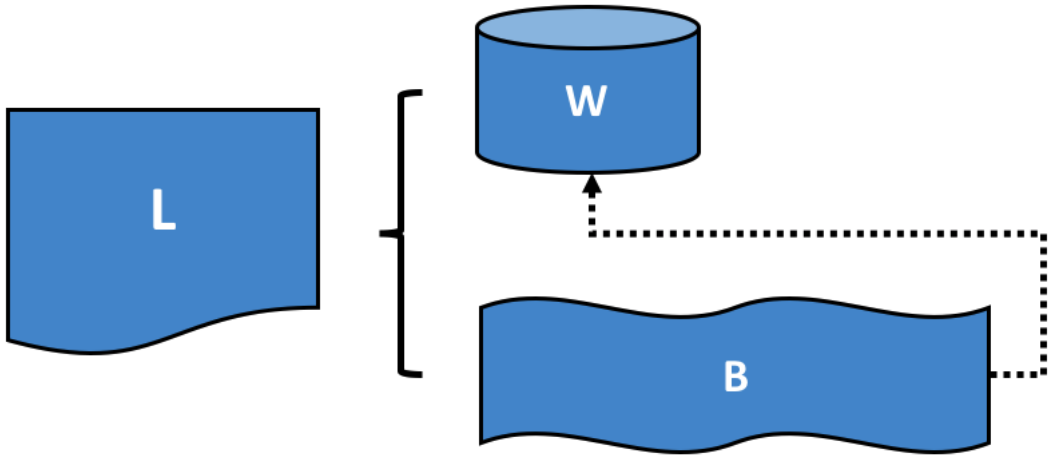
② <TRANSACTION-ENDORSED, tid, tran-proposal,epSig>

- tran-proposal := (epID,tid,chaincodeID,txContentBlob,readset,writeset)



ledger

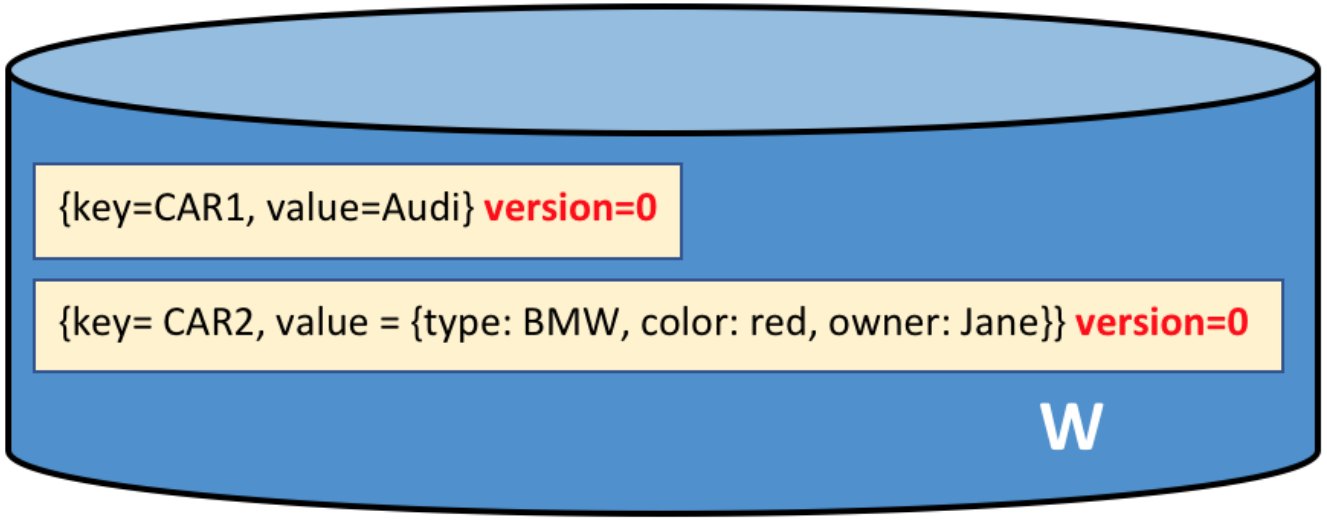
- A ledger consists of: a world state and a blockchain
- A world state
  - Current values of a set of ledger states
  - Expressed as key-value pairs, which are created, updated, deleted
- A blockchain
  - A TX log that records all the changes that have resulted in the world state
  - immutable



L	Ledger
W	World State
B	Blockchain
L: B, W	L comprises B and W
W ◄ B	B determines W

World state (1/2)

- Current value of the attributes of a biz object



W	Ledger world state
{key=**K**, value = **V** } **version=0**	A ledger state with **key=K**. It contains a set of facts expressed as a simple value, **V**. The state is at version 0.
{key=**K**, value = {**KV**} } **version=0**	A ledger state with **key=K**. It contains a set of facts expressed as a set of key-value pairs **{KV}.** The state is at version 0.

A ledger world state containing two states. The first state is: key=CAR1 and value=Audi. The second state has a more complex value: key=CAR2 and value={model:BMW, color=red, owner=Jane}. Both states are at version 0.

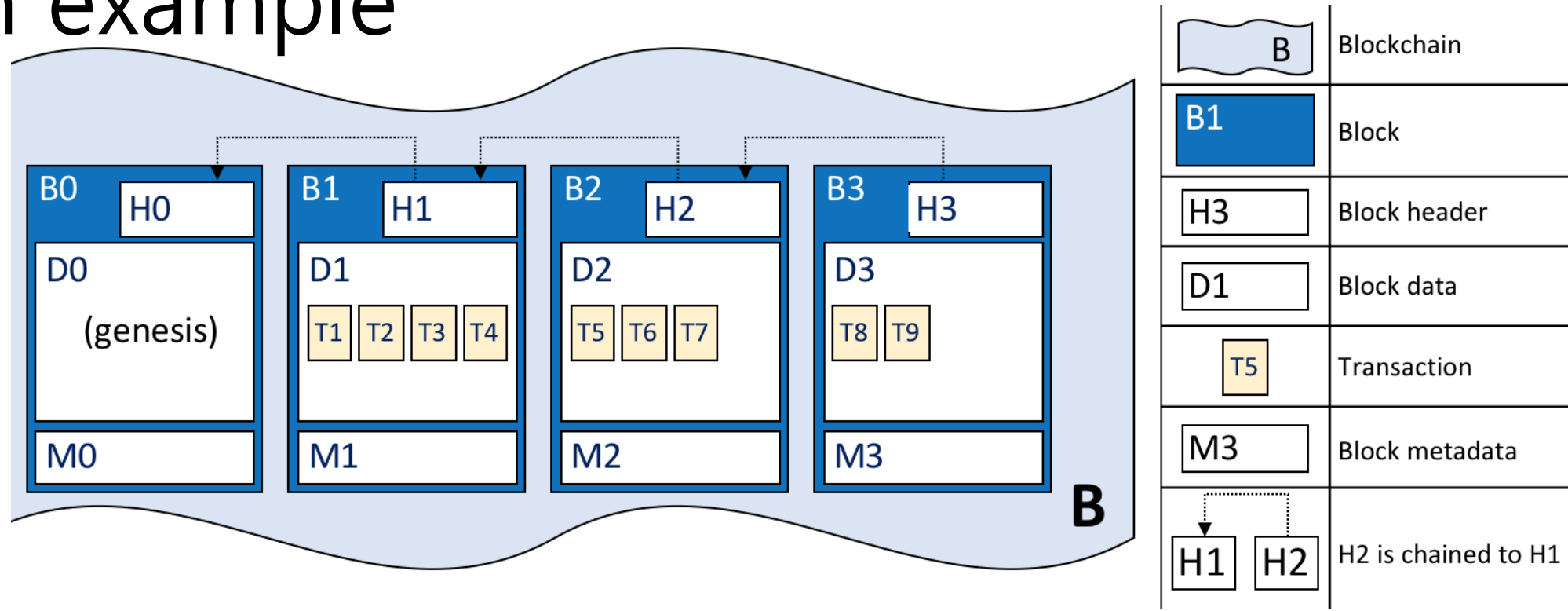
World state (2/2)

- An application program can invoke a smart contract which uses simple ledger APIs to **get**, **put** and **delete** states
- Implemented as a database
- only transactions that are **signed** by the required set of **endorsing organizations** will result in an update to the world state
- version number: for internal use by Hyperledger Fabric, and incremented every time the state changes
- world state can be re-generated from the blockchain at any time
  - If a node fails and restarts

blockchain

- Has recorded every previous version of each ledger state and how it has been changed
- sequential log of interlinked blocks
  - each block contains a sequence of TXs: ordering service
- Each TX represents a query or update to the world state
- implemented as a file

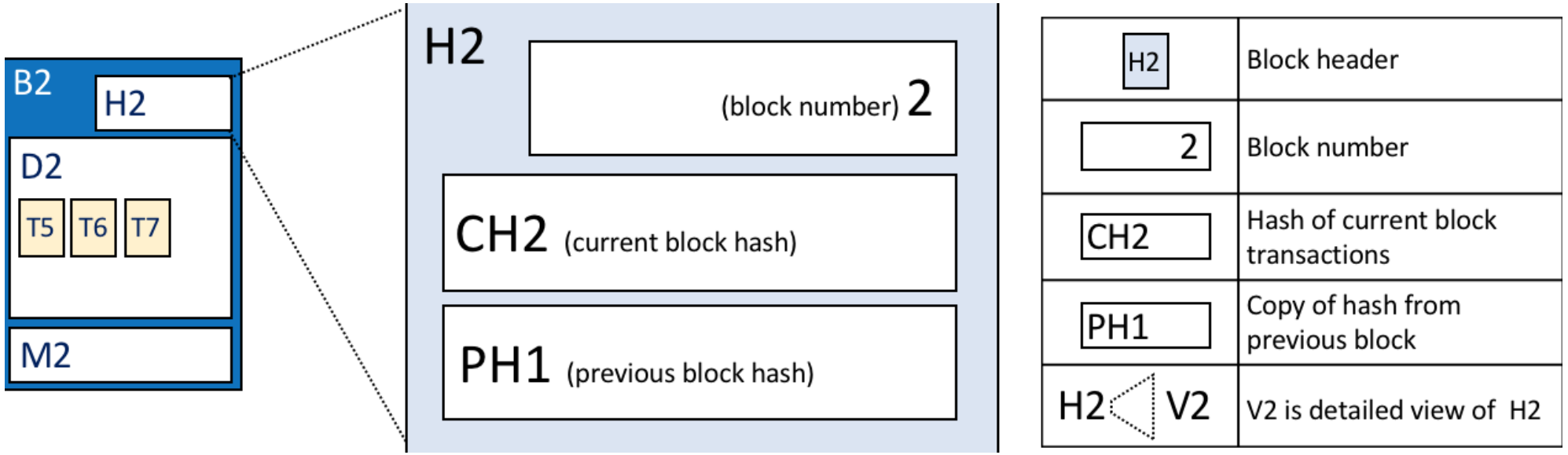
Blockchain example



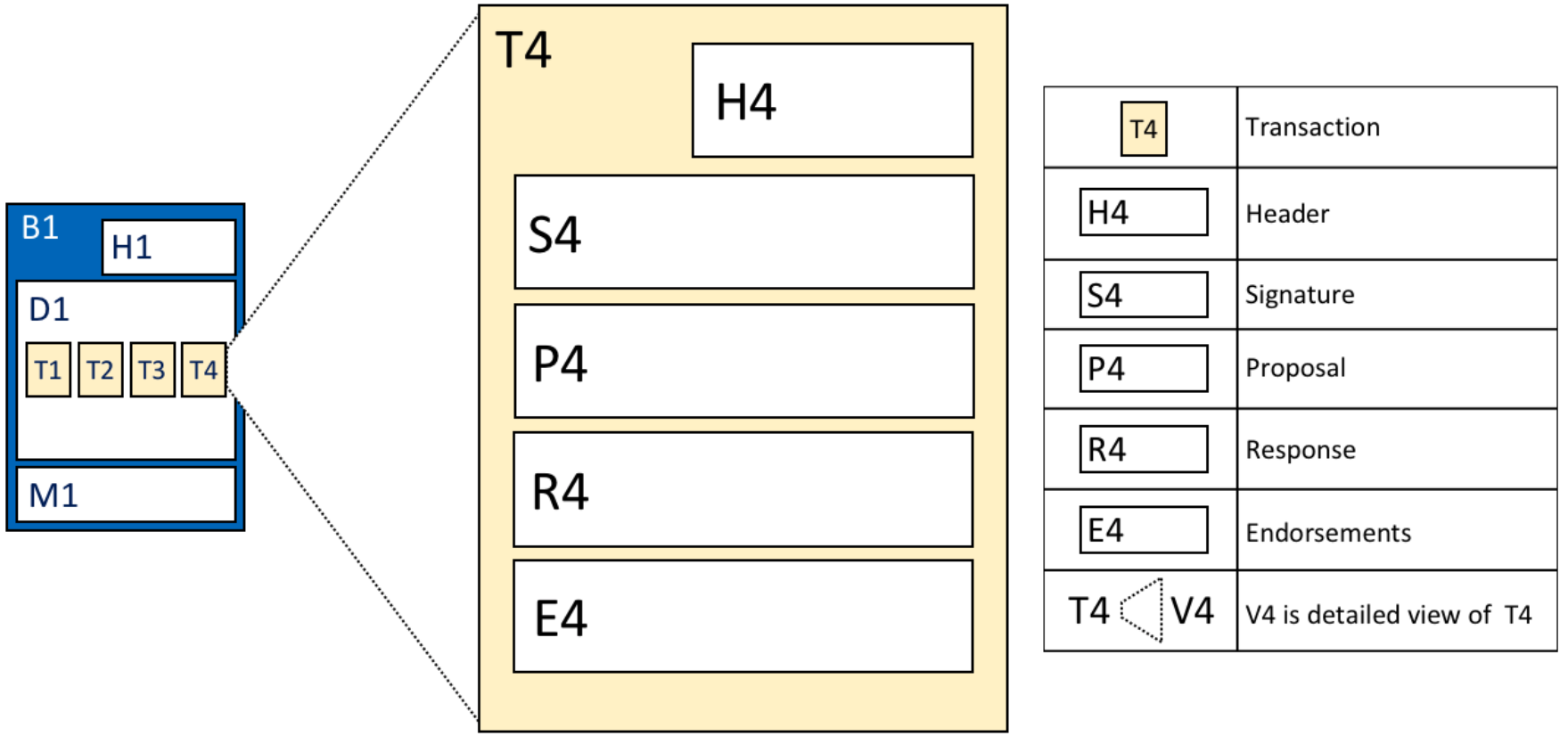
- A blockchain B containing 4 blocks; B0 is the genesis block in the blockchain
- We can see that **block** B2 has a **block data** D2 which contains all its TXs: T5, T6, T7.
- B2 has a **block header** H2, which contains a cryptographic **hash** of all the TXs in D2 as well as with the equivalent hash from the previous block B1. In this way, blocks are inextricably and immutably linked to each other
- Genesis block has a configuration TX containing the initial state of the network

blocks

- Block header
  - Block number: An integer starting at 0 (the genesis block), and increased by 1 for every new block appended to the blockchain.
  - Current Block Hash and Previous Block Hash
- Block data: a list of TXs
- Block metadata
  - Block generation time
  - Certificate and signature of the block writer (orderer)
  - Block committer (committing peer) adds a valid/invalid indicator



TXs



- Header: metadata about the TX; name and version of chaincode
- Signature by client application
- Proposal: inputs by an app to the smart contract
- Response: Read Write set, output of a smart contract
- Endorsements: a list of signed TX responses (from endorsers)